

```
import kagglehub
chrisfilo_urbansound8k_path = kagglehub.dataset_download('chrisfilo/urbansound8k')

print('Data source import complete.')
```

Downloading from https://www.kaggle.com/api/v1/datasets/download/chrisfilo/urbansound8k?dataset_version_number=1...
100%|██████████| 5.61G/5.61G [01:27<00:00, 69.1MB/s]Extracting files...

Data source import complete.

✓ Κατηγοριοποίηση Ήχου με CNN

Αυτό το project δείχνει πώς να ταξινομήσετε ηχητικά αρχεία χρησιμοποιώντας τεχνικές βαθιάς μάθησης, συγκεκριμένα εξάγοντας χαρακτηριστικά από αρχεία ήχου και εκπαιδεύοντας ένα Συνελικτικό Νευρωνικό Δίκτυο (CNN).

✓ Φόρτωση αναγκαιών βιβλιοθηκών

```
import os
import librosa
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
import warnings
warnings.filterwarnings('ignore')
```

✓ Φόρτωση του Συνόλου Δεδομένων

Αυτό το βήμα διαβάζει τα metadata του συνόλου δεδομένων UrbanSound8K από ένα αρχείο CSV σε ένα DataFrame της Pandas. Τα metadata περιλαμβάνουν πληροφορίες για τα αρχεία ήχου, όπως τις ετικέτες κλάσεων τους και τις διαδρομές των αρχείων.

```
metadata = pd.read_csv('/root/.cache/kagglehub/datasets/chrisfilo/urbansound8k/versions/1/UrbanSound8K.csv')
```

```
metadata.head()
```

	slice_file_name	fsID	start	end	salience	fold	classID	class
0	100032-3-0-0.wav	100032	0.0	0.317551	1	5	3	dog_bark
1	100263-2-0-117.wav	100263	58.5	62.500000	1	5	2	children_playing
2	100263-2-0-121.wav	100263	60.5	64.500000	1	5	2	children_playing
3	100263-2-0-126.wav	100263	63.0	67.000000	1	5	2	children_playing
4	100263-2-0-137.wav	100263	68.5	72.500000	1	5	2	children_playing

Επόμενα βήματα:

[Δημιουργία κώδικα με metadata](#)

[Προβολή προτεινόμενων γραφημάτων](#)

[New interactive sheet](#)

✓ Εξαγωγή χαρακτηριστικών από τα αρχεία ήχου

```
def extract_features(file_path):
    y, sr = librosa.load(file_path, duration=4.0)
    n_fft = min(2048, len(y) // 2)
    mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=40, n_fft=n_fft)
    mfcc_scaled = np.mean(mfcc.T, axis=0)
    return mfcc_scaled
```

Η συνάρτηση `extract_features` φορτώνει ένα αρχείο ήχου, εξάγει τα χαρακτηριστικά MFCC του, και επιστρέφει τον μέσο όρο των MFCC στο χρόνο. Τα MFCC χρησιμοποιούνται συχνά στην επεξεργασία ήχου για να αναπαράσθουν το βραχυπρόθεσμο φάσμα ισχύος του ήχου.

```
def prepare_dataset(metadata, dataset_path):
    features = []
    labels = []

    for i, row in metadata.iterrows():
        file_name = os.path.join(dataset_path, 'fold' + str(row["fold"]), str(row["slice_file_name"]))
        class_label = row["class"]
        try:
            mfccs = extract_features(file_name)
            features.append(mfccs)
            labels.append(class_label)
        except Exception as e:
            print(f"Error loading {file_name}: {e}")

    return np.array(features), np.array(labels)
```

- Η συνάρτηση `prepare_dataset` διατρέχει κάθε γραμμή στα `metadata`, κατασκευάζει τη διαδρομή αρχείου για κάθε αρχείο ήχου, και εξάγει τα χαρακτηριστικά του χρησιμοποιώντας την προηγουμένως ορισμένη συνάρτηση `extract_features`. Τα χαρακτηριστικά και οι αντίστοιχες ετικέτες αποθηκεύονται σε λίστες και επιστρέφονται ως πίνακες NumPy.

✓ Φόρτωση και Προεπεξεργασία Συνόλου Δεδομένων

```
dataset_path = '/root/.cache/kagglehub/datasets/chrisfilo/urbansound8k/versions/1'
```

```
X, y = prepare_dataset(metadata, dataset_path)
```

Αυτός ο κώδικας καθορίζει τη διαδρομή του συνόλου δεδομένων, στη συνέχεια καλεί τη συνάρτηση `prepare_dataset` για να φορτώσει τα δεδομένα ήχου και να εξάγει τα χαρακτηριστικά και τις ετικέτες. Τα χαρακτηριστικά αποθηκεύονται στο `x`, και οι ετικέτες αποθηκεύονται στο `y`.

✓ Κωδικοποίηση Ετικετών

```
le = LabelEncoder()
y_encoded = to_categorical(le.fit_transform(y))
```

Εδώ, οι ετικέτες κωδικοποιούνται σε ακέραιους χρησιμοποιώντας το `LabelEncoder`, και στη συνέχεια μετατρέπονται σε κωδικοποίηση `one-hot` χρησιμοποιώντας το `to_categorical`. Αυτό είναι απαραίτητο για εργασίες κατηγοριοποίησης όπου κάθε κλάση αναπαρίσταται από ένα δυαδικό διάνυσμα.

```
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)
```

Το σύνολο δεδομένων χωρίζεται σε σύνολα εκπαίδευσης και δοκιμών, με το 80% των δεδομένων να χρησιμοποιείται για εκπαίδευση και το 20% για δοκιμές. Αυτό είναι ένα κρίσιμο βήμα για την αξιολόγηση της απόδοσης του μοντέλου.

```
print("Original X_train shape:", X_train.shape)
```

```
➦ Original X_train shape: (6985, 40)
```

✓ Αναδιαμόρφωση Δεδομένων Εισόδου για CNN

```
num_features = X.shape[1]
X_train = X_train.reshape(X_train.shape[0], 40, 1, 1)
X_test = X_test.reshape(X_test.shape[0], 40, 1, 1)
```

Τα δεδομένα εισόδου αναδιαμορφώνονται για να ταιριάζουν στις απαιτήσεις εισόδου του CNN. Κάθε δείγμα θα έχει διαστάσεις που αντιστοιχούν στον αριθμό των χαρακτηριστικών MFCC, το ύψος και τα κανάλια.

```
print("Reshaped X_train shape:", X_train.shape)
print("Reshaped X_test shape:", X_test.shape)
```

```
➦ Reshaped X_train shape: (6985, 40, 1, 1)
Reshaped X_test shape: (1747, 40, 1, 1)
```

✓ Κατασκευή του Μοντέλου CNN

```
model = Sequential()

model.add(Conv2D(32, kernel_size=(2, 1), activation='relu', input_shape=(40, 1, 1)))
model.add(MaxPooling2D(pool_size=(2, 1)))
model.add(Dropout(0.25))

model.add(Conv2D(64, kernel_size=(2, 1), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 1)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(y_encoded.shape[1], activation='softmax'))

model.summary()
```

→ Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 39, 1, 32)	96
max_pooling2d (MaxPooling2D)	(None, 19, 1, 32)	0
dropout (Dropout)	(None, 19, 1, 32)	0
conv2d_1 (Conv2D)	(None, 18, 1, 64)	4,160
max_pooling2d_1 (MaxPooling2D)	(None, 9, 1, 64)	0
dropout_1 (Dropout)	(None, 9, 1, 64)	0
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 128)	73,856
dropout_2 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

Total params: 79,402 (310.16 KB)
Trainable params: 79,402 (310.16 KB)

Αυτή η ενότητα κατασκευάζει ένα Συνελκτικό Νευρωνικό Δίκτυο (CNN) χρησιμοποιώντας το Keras Sequential API. Το μοντέλο αποτελείται από:

- Επίπεδα Conv2D: για εξαγωγή χαρακτηριστικών από τα δεδομένα εισόδου.
- Επίπεδα MaxPooling2D: για μείωση του dimensionality και έλεγχο του overfitting.
- Επίπεδα Dropout: για την πρόληψη του overfitting με τυχαία απόρριψη μονάδων κατά την εκπαίδευση.
- Επίπεδο Flatten: για τη μετατροπή των 2D χαρτών χαρακτηριστικών σε ένα 1D διάνυσμα χαρακτηριστικών.
- Επίπεδα Dense: για ταξινόμηση με ενεργοποίηση softmax στο επίπεδο εξόδου για την πρόβλεψη των πιθανοτήτων των κλάσεων.

✓ Compile το μοντέλο

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Το μοντέλο κάνει compile με τη συνάρτηση απώλειας categorical cross-entropy, η οποία είναι κατάλληλη για προβλήματα ταξινόμησης πολλαπλών κλάσεων. Ο βελτιστοποιητής Adam χρησιμοποιείται για αποτελεσματική εκπαίδευση, και η ακρίβεια καθορίζεται ως μετρική για παρακολούθηση.

✓ Εκπαίδευση του μοντέλου

```
model.fit(X_train, y_train, epochs=30, batch_size=32, validation_data=(X_test, y_test))
```

→ 219/219 ————— 2s 9ms/step - accuracy: 0.2570 - loss: 2.0728 - val_accuracy: 0.4533 - val_loss: 1.7351
Epoch 3/30
219/219 ————— 2s 9ms/step - accuracy: 0.3111 - loss: 1.9265 - val_accuracy: 0.4820 - val_loss: 1.5470
Epoch 4/30

```

Epoch 6/30
219/219 ————— 3s 15ms/step - accuracy: 0.4212 - loss: 1.6302 - val_accuracy: 0.5821 - val_loss: 1.3089
Epoch 7/30
219/219 ————— 2s 8ms/step - accuracy: 0.4498 - loss: 1.5779 - val_accuracy: 0.6062 - val_loss: 1.2031
Epoch 8/30
219/219 ————— 3s 8ms/step - accuracy: 0.4748 - loss: 1.5140 - val_accuracy: 0.6285 - val_loss: 1.1370
Epoch 9/30
219/219 ————— 2s 8ms/step - accuracy: 0.4901 - loss: 1.4550 - val_accuracy: 0.6422 - val_loss: 1.1207
Epoch 10/30
219/219 ————— 3s 9ms/step - accuracy: 0.5079 - loss: 1.4324 - val_accuracy: 0.6474 - val_loss: 1.0731
Epoch 11/30
219/219 ————— 3s 13ms/step - accuracy: 0.5415 - loss: 1.3428 - val_accuracy: 0.6823 - val_loss: 1.0487
Epoch 12/30
219/219 ————— 4s 10ms/step - accuracy: 0.5314 - loss: 1.3374 - val_accuracy: 0.7018 - val_loss: 0.9841
Epoch 13/30
219/219 ————— 3s 10ms/step - accuracy: 0.5664 - loss: 1.2601 - val_accuracy: 0.7041 - val_loss: 0.9558
Epoch 14/30
219/219 ————— 3s 10ms/step - accuracy: 0.5775 - loss: 1.2521 - val_accuracy: 0.7218 - val_loss: 0.9363
Epoch 15/30
219/219 ————— 3s 12ms/step - accuracy: 0.5892 - loss: 1.2068 - val_accuracy: 0.7293 - val_loss: 0.8936
Epoch 16/30
219/219 ————— 4s 8ms/step - accuracy: 0.5930 - loss: 1.1572 - val_accuracy: 0.7418 - val_loss: 0.8758
Epoch 17/30
219/219 ————— 3s 9ms/step - accuracy: 0.5969 - loss: 1.2130 - val_accuracy: 0.7516 - val_loss: 0.8588
Epoch 18/30
219/219 ————— 2s 8ms/step - accuracy: 0.6159 - loss: 1.1191 - val_accuracy: 0.7773 - val_loss: 0.7950
Epoch 19/30
219/219 ————— 3s 10ms/step - accuracy: 0.6157 - loss: 1.1506 - val_accuracy: 0.7687 - val_loss: 0.8054
Epoch 20/30
219/219 ————— 3s 13ms/step - accuracy: 0.6310 - loss: 1.0686 - val_accuracy: 0.7728 - val_loss: 0.7716
Epoch 21/30
219/219 ————— 2s 8ms/step - accuracy: 0.6217 - loss: 1.0915 - val_accuracy: 0.7790 - val_loss: 0.7516
Epoch 22/30
219/219 ————— 2s 8ms/step - accuracy: 0.6278 - loss: 1.0951 - val_accuracy: 0.7871 - val_loss: 0.7517
Epoch 23/30
219/219 ————— 3s 9ms/step - accuracy: 0.6407 - loss: 1.0342 - val_accuracy: 0.7842 - val_loss: 0.7020
Epoch 24/30
219/219 ————— 2s 9ms/step - accuracy: 0.6464 - loss: 1.0015 - val_accuracy: 0.7956 - val_loss: 0.7073
Epoch 25/30
219/219 ————— 3s 12ms/step - accuracy: 0.6618 - loss: 0.9959 - val_accuracy: 0.7899 - val_loss: 0.6781
Epoch 26/30
219/219 ————— 4s 8ms/step - accuracy: 0.6680 - loss: 0.9947 - val_accuracy: 0.8019 - val_loss: 0.6856
Epoch 27/30
219/219 ————— 3s 9ms/step - accuracy: 0.6810 - loss: 0.9288 - val_accuracy: 0.8082 - val_loss: 0.6288
Epoch 28/30
219/219 ————— 3s 9ms/step - accuracy: 0.6685 - loss: 0.9649 - val_accuracy: 0.8077 - val_loss: 0.6429
Epoch 29/30
219/219 ————— 2s 10ms/step - accuracy: 0.6786 - loss: 0.9323 - val_accuracy: 0.8174 - val_loss: 0.6231
Epoch 30/30
219/219 ————— 3s 12ms/step - accuracy: 0.6908 - loss: 0.8990 - val_accuracy: 0.8151 - val_loss: 0.5951
<keras.src.callbacks.history.History at 0x78394871d450>

```

Το μοντέλο εκπαιδεύεται στα δεδομένα εκπαίδευσης για 30 εποχές με μέγεθος παρτίδας 32. Τα δεδομένα επικύρωσης παρέχονται για την παρακολούθηση της απόδοσης του μοντέλου σε άγνωστα δεδομένα μετά από κάθε εποχή.

✓ Απόδοση μοντέλου

```

score = model.evaluate(X_test, y_test, verbose=0)
print(f'Test accuracy: {score[1]*100:.2f}%')

```

```

↗ Test accuracy: 81.51%

```

```

y_pred = model.predict(X_test)
y_test_labels = le.inverse_transform(np.argmax(y_test, axis=1))
y_pred_labels = le.inverse_transform(np.argmax(y_pred, axis=1))
print(classification_report(y_test_labels, y_pred_labels))

```

```

↗ 55/55 ————— 1s 15ms/step
              precision    recall  f1-score   support

air_conditioner    0.81      0.93      0.86       203
  car_horn         0.90      0.83      0.86        86
children_playing   0.61      0.77      0.68       183
  dog_bark         0.84      0.73      0.78       201
  drilling         0.80      0.77      0.78       206
engine_idling      0.89      0.94      0.92       193
  gun_shot         0.90      0.75      0.82        72
jackhammer         0.91      0.90      0.91       208
  siren            0.84      0.95      0.89       165
street_music       0.80      0.60      0.68       230

               accuracy                   0.82       1747
            macro avg      0.83      0.82      0.82       1747
           weighted avg      0.82      0.82      0.81       1747

```

✓ Δοκιμή Μοντέλου

```
# Συνάρτηση για να φορτώσει ένα αρχείο ήχου και να εξάγει τα χαρακτηριστικά MFCC
def load_and_prepare_audio(file_path):
    mfccs = extract_features(file_path) # Εξαγωγή χαρακτηριστικών MFCC
    mfccs = mfccs.reshape(1, 40, 1, 1) # Αναδιάταξη για το μοντέλο
    return mfccs


# Διαδρομή του αρχείου ήχου
audio_file_path = '/root/.cache/kagglehub/datasets/chrisfilo/urbansound8k/versions/1/fold1/102305-6-0-0.wav' # Αντικαταστήστε με τη δική σας

# Φόρτωση και προετοιμασία του ήχου
audio_features = load_and_prepare_audio(audio_file_path)

# Προβλέψεις
predictions = model.predict(audio_features)
predicted_class = np.argmax(predictions, axis=1) # Πάρε το δείκτη της κλάσης με τη μεγαλύτερη πιθανότητα


# Αποκωδικοποίηση της προβλεπόμενης κλάσης
predicted_label = le.inverse_transform(predicted_class) # Μετατροπή του δείκτη πίσω στην αρχική ετικέτα της κλάσης

print(f"Η προβλεπόμενη ετικέτα για τον ήχο είναι: {predicted_label[0]}")
```

 1/1 ————— 0s 64ms/step
Η προβλεπόμενη ετικέτα για τον ήχο είναι: children_playing

✓ Αποθήκευση

```
# Αποθήκευση μοντέλου
model.save('urban_sound_model.h5')

 WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
< ————— >

# Αποθήκευση κλάσεων
np.save('classes.npy', le.classes_)
```